MP-Flow: A Deep Graph Reinforcement Learning Agent for Maximizing Throughput in the Lightning Network

Harrison Rush
Amboss Technologies
Green Cove Springs, USA
harrison@amboss.tech

Vincent
Amboss Technologies
Green Cove Springs, USA
v@amboss.tech

Vikash Singh
Stillmark
San Francisco, USA
vikash@stillmark.com

Jesse Shrader
Amboss Technologies
Green Cove Springs, USA
j@amboss.tech

Emanuele Rossi
Amboss Technologies
Barcelona, Spain
emanuele.rossi1909@gmail.com

Abstract

We address the challenge of reliable payment routing in the Bitcoin Lightning Network (LN), an emerging technology designed to scale Bitcoin transactions through off-chain channels. A key bottleneck to LN performance is liquidity placement, ensuring sufficient channel capacity along payment paths to maximize throughput. We formulate throughput-oriented liquidity placement as a graph reinforcement learning problem and introduce a lightweight agent that combines a message-passing policy network with proximal policy optimization (PPO) and action masking for stable and generalizable learning. To provide a theory-grounded yet scalable training reward, we use max flow as a proxy for payment throughput. In extensive experiments on real Lightning Network snapshots, our method consistently outperforms strong heuristic baselines across multiple seeds and unseen graphs. The agent has been deployed in production for peer recommendations, facilitating over $5,000,000 in BTC in liquidity allocation across 200 channels. Our results highlight the potential of graph-based reinforcement learning for adaptive resource allocation in decentralized financial systems.

1 Introduction

Payment-channel networks such as the Bitcoin Lightning Network (LN) enable fast, low-cost payments by moving transactions off-chain. Their performance, however, hinges on *where* liquidity is placed: poor placement creates bottlenecks that throttle routing capacity, while targeted placement can unlock throughput. For LN operators the question is simple to state and hard to solve: *which peers should I connect to, and how should I allocate scarce liquidity to participate effectively?* Existing practice leans on static graph heuristics (e.g., degree or betweenness) that ignore directed balances, budgeted interventions, and interactions among multiple allocations [16, 5].

The LN offers little external observability and no supervised traces; realistic traffic simulation demands strong assumptions about demand matrices, retry logic, and hidden balances that can dominate conclusions (see, e.g., LN measurement/topology studies) [23, 21]. We therefore frame throughput-oriented liquidity placement as a graph RL problem and adopt a theory-grounded *proxy* for end-to-end capacity: *maximum s–t flow* on the observed capacity graph [4, 1]. Improving cut bottlenecks improves feasible routing, while avoiding brittle and expensive traffic simulation.

We instantiate a simple, deployable policy that combines a message-passing neural network (MPNN) with an on-policy optimizer (PPO) and feasibility-aware action masking. The MPNN uses a permutation-invariant *max* aggregator to emphasize narrow-cut structure in heterogeneous-capacity neighborhoods [6], and PPO provides stable, sample-efficient updates without second-order steps [22].

Preprint. Preliminary work.

We evaluate under a paired protocol on real LN snapshots and report both *seen-graph* performance and *generalization to unseen snapshots*.

Contributions.

- **Outcome-aligned RL formulation:** liquidity placement as a graph decision process with reward defined by *relative improvement in Max-Flow* on observed capacity graphs [1].
- **Simple, deployable policy:** a compact MPNN–PPO agent with action masking and max aggregation [6, 22].
- **Empirical evidence on real LN topology:** consistent improvements over strong heuristic baselines under paired evaluation, with robustness checks and multi-seed, multi-snapshot results (cf. LN topology variability [23]).
- **Practical relevance:** the approach has been *deployed in practice* to inform peer recommendations.

The remainder of the paper introduces the necessary background on Max-Flow, RL, and MPNNs (§2); explains how we assemble these standard components for LN liquidity placement and why Max-Flow is an appropriate proxy in this setting (§4); details datasets, metrics, and protocols (§5); and presents results on seen and unseen graphs with robustness analyses (§6). We discuss limitations of proxy-based evaluation and pathways toward richer traffic models in future work.

2 Background

2.1 Bitcoin & Lightning Network

Bitcoin is a decentralized digital currency introduced by Satoshi Nakamoto in 2008 [15], operating without intermediaries like banks or governments. It uses blockchain technology—distributed ledger consisting of blocks that contain transaction data. Each block is cryptographically linked to the previous one, forming a chain that prevents tampering and double-spending. However, Bitcoin faces scalability issues, processing approximately seven transactions per second with confirmation times ranging from 10 minutes to an hour. High transaction fees during network congestion further limit its usability for everyday transactions. The Lightning Network (LN) is a second-layer protocol for Bitcoin that enables fast, low-cost transactions by using off-chain payment channels [20]. Channels lock funds in a multi-signature address and allow participants to exchange updates privately; only the opening and closing transactions are recorded on-chain. Linking many such channels creates a peer-to-peer network that can route payments between any two nodes. Each channel has a fixed capacity, so insufficient liquidity may block routes. Because the network's topology and flows change continuously, nodes must actively manage liquidity to maintain performance.

2.2 Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a class of neural network architectures specifically designed to process data represented as graphs, effectively capturing the complex dependencies and relationships inherent in such structures [10]. Graphs are mathematical structures consisting of nodes (vertices) V and edges E, denoted as $G = (V, E)$, where edges represent connections or interactions between nodes. In many real-world applications, data naturally resides in non-Euclidean domains with irregular structures, such as social networks, biological networks, and communication networks like the Lightning Network.

Let $h_v^{(\ell)}$ denote the embedding of node v at layer ℓ, and e_{uv} edge features on (u, v). A generic Message Passing Neural Network (MPNN) layer is:

$$m_v^{(\ell)} = \square_{u \in \mathcal{N}(v)} \psi^{(\ell)}\big(h_u^{(\ell)},\, h_v^{(\ell)},\, e_{uv}\big), \qquad h_v^{(\ell+1)} = \phi^{(\ell)}\big(h_v^{(\ell)},\, m_v^{(\ell)}\big),$$

where $\psi^{(\ell)}$ and $\phi^{(\ell)}$ are learnable functions and $\square$ is a permutation-invariant aggregator (e.g., SUM/MEAN/MAX) [7].

2.3 Max-Flow

We consider a directed graph $G = (V, E)$ with capacities $c : E \to \mathbb{R}_{\geq 0}$, source $s \in V$, and sink $t \in V$. A flow is a function $f : E \to \mathbb{R}_{\geq 0}$ satisfying capacity constraints $0 \leq f(u, v) \leq c(u, v)$

and flow conservation $\sum_u f(u,v) = \sum_w f(v,w)$ for all $v \in V \setminus \{s,t\}$. The *value* of the flow is $|f| = \sum_w f(s,w) = \sum_u f(u,t)$. The *maximum flow* problem seeks $f^\star$ maximizing $|f|$. By the Max-Flow Min-Cut theorem, the maximum flow equals the minimum s–t cut capacity. Classical algorithms include augmenting paths (Ford–Fulkerson / Edmonds–Karp) and push–relabel variants.

2.4 Reinforcement Learning (MDP/PPO)

An episodic Markov Decision Process (MDP) is $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition kernel P, reward r, and discount γ. Policy-gradient methods optimize $J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_t \gamma^t r_t]$. Proximal Policy Optimization (PPO) maximizes a clipped surrogate objective using on-policy trajectories with an advantage estimator; see [22] for details.

3 Related Work

In this section, we review the existing literature on optimizing the Lightning Network, maxflow algorithms in payment networks, and the application of deep graph reinforcement learning. $LightningNetworkDaemon$ (LND) one of the most widely used implementations of the Lightning Network protocol developed by Lightning Labs, includes an Autopilot feature. This module automatically opens channels on behalf of the user based on certain heuristics such as degree and betweenness centrality [**?**]. This demonstrates an early approach to automating node and channel management. However, it relies on predefined heuristics rather than learning-based methods. Other Lightning Network implementations, such as Blockstream's C-lightning and ACINQ's Eclair, have explored autopilot functionalities through plugins and external tools. These implementations often allow for customization and integration with third-party autopilot solutions [3]. These tools allow for automated channel selection, but still require significant domain knowledge and custom setup. Beyond community plugins, Lightning Labs' production tooling (Lightning Terminal "AutoOpen") explicitly uses betweenness centrality to score new peers, reflecting current industry practice for channel selection heuristics [11, 12]. This motivates our choice of Betweenness as the primary baseline for practical comparisons.

Pickhardt and Richter model path reliability by treating channel balances as random variables and computing most-likely, low-cost multi-part payments via (generalized) min-cost flow; they also argue for zero base fees to linearize the objective [18]. Complementary analyses formalize payment success under uncertain balances and derive bounds/estimators for end-to-end success [19, 17]. Closer to our data model, Davis et al. propose *channel balance interpolation* (CBI), predicting local/remote splits from node/channel features; such priors can reduce variance in simulation and improve routing objectives [25, 2]. Our work differs by optimizing *absolute max-flow uplift* with a learned MPNN–PPO policy, but shares the premise that directed balances and bottlenecks—rather than raw centrality—govern deliverability.

Wang et al. [26] proposed Flash, a dynamic routing algorithm designed to improve transaction success rates by optimizing liquidity utilization across the network. Their approach uses modified maxflow for large payments combined with a routing table lookup for small payments. Sivaraman et al. [24] proposed the Spider Network, a high-throughput routing solution employing a congestion control algorithm inspired by maxflow optimization. Recent advancements in graph neural networks (GNNs) have introduced novel strategies for maximizing throughput in complex network deployments, leveraging reinforcement learning to iteratively adjust node placements in alignment with throughput optimization goals, as demonstrated by Yang et al. [27] in their GNN-based PPO framework.

Despite advancements, there remains a gap in integrating these approaches to address liquidity allocation specifically. Previous studies have focused on routing algorithms, security vulnerabilities, and general network optimization but have not leveraged deep reinforcement learning with graph neural networks to optimize liquidity distribution in the Lightning Network.

4 Method

4.1 Lightning Network as a Graph

We model the public LN snapshot as a directed, weighted graph $G = (V, E)$. Each undirected channel $\{u, v\}$ yields two directed edges (u, v) and (v, u) whose capacities represent local balances;

only the total capacity is observed, so per-direction balances are sampled once per run. Node features (PageRank, capacity ratio, normalized degree, clustering) and optional edge features (base fee, fee rate, channel capacity) are z-scored per reset; see App E.0.1 for definitions and transforms.

4.2 Feature Engineering

At each environment reset we construct a node–feature matrix $X \in \mathbb{R}^{N\times 4}$ and, when enabled, an edge–feature matrix $E \in \mathbb{R}^{M\times 3}$. The node features are: (i) PageRank centrality $C_{PR}(u)$ for global influence; (ii) capacity ratio $C_{CR}(u) = C_u / \sum_{v\in V} C_v$ for a node's share of network liquidity; (iii) normalized degree $C_D(u) = \deg(u)/(N-1)$ for relative connectivity; and (iv) local clustering coefficient $C_C(u)$ for neighborhood density. When `use_edge_features` is active, each directed edge (u, v) provides three attributes: base fee f_{uv}^{base}, proportional fee rate f_{uv}^{rate}, and channel capacity c_{uv}. To mitigate heavy tails, edge attributes undergo a $\log(1 + x)$ transform and per-column standardization $E \leftarrow (\tilde{E} - \mu_E) \oslash (\sigma_E + \varepsilon)$; missing fee fields default to zero so E maintains shape $M \times 3$. All features are recomputed after any topological change and z-scored per dimension before being consumed by the MPNN.

4.3 Markov Decision Process

We use a finite-horizon MDP with k=5 actions. The state encodes G_t and features (X, E); actions are masked node picks (open/top-up) under a per-action budget. Transitions deterministically update capacities. The per-step reward is the *marginal* improvement in absolute max-flow,

$$r_t = F_t - F_{t-1},$$

where F_t is computed via push–relabel on the a sample of random targets in the network and summed. Further implementation details are in App E.

Policy and Learning Algorithm

We instantiate the policy as a message-passing GNN (see §2) with a permutation-invariant **max** aggregator to better highlight narrow cut bottlenecks. Node and edge features feed the message and update functions; a small readout produces action logits over the admissible set, with feasibility enforced by *masked* sampling. For learning, we adopt a standard on-policy objective (PPO) to couple stable updates with straightforward deployment. The specific message/update parameterization, action masking, and PPO settings are reported in §5; extended diagrams appear in App. A.

4.4 Algorithm Selection

We evaluated A2C [14], PPO [22], DDPG [13], and a custom hybrid ("GCN2") with discrete node picks plus continuous allocations. Off-policy/continuous variants (DDPG/GCN2) were brittle and failed to converge reliably under our reward and topology shifts; A2C learned but exhibited instability. PPO's clipped surrogate and entropy regularization delivered stable, sample-efficient learning with the masked discrete action space, so we adopt PPO and vary only the function approximator (MPNN-Max).

4.5 Application-specific modifications.

We modify the standard MPNN–PPO framework to operate directly on the Lightning Network (LN) topology, where each edge encodes a bidirectional payment channel with known capacity and policy attributes. The agent's objective is to allocate liquidity over visible nodes to maximize network throughput, measured as the marginal change in global max-flow.

Message-passing backbone. Each node i is initialized with a feature vector $h_i^{(0)}$ containing its local degree, capacity share, and policy metadata. At each message-passing layer l, node i aggregates messages from its neighbors $\mathcal{N}(i)$:

$$m_{ij}^{(l)} = \text{MLP}_m^{(l)}\big(h_i^{(l)} \,\|\, h_j^{(l)} \,\|\, e_{ij}\big), \qquad e_{ij} = [c_{ij},\, \text{age}_{ij},\, \text{fee}_{ij}], \tag{1}$$

where e_{ij} encodes channel capacity, policy, and topological cues. Instead of mean or sum aggregation, we employ a *max* operator to emphasize the single most constraining neighbor—mimicking bottleneck detection along min-cut edges:

$$h_i^{(l+1)} = \mathrm{MLP}_h^{(l)}\Big(h_i^{(l)} \,\|\, \max_{j\in\mathcal{N}(i)} m_{ij}^{(l)}\Big). \tag{2}$$

This **max aggregation** (Eq. 2) replaces the usual mean-pooling used in standard MPNNs and allows the policy to focus on locally binding capacity constraints rather than diffuse averages.

Actor–critic heads. After L message-passing layers, we obtain node embeddings $h_i^{(L)}$. Global representations $f_\theta(G)$ are computed by pooling across visible nodes, and the shared backbone feeds two separate output heads:

$$\pi_\theta(a_t|s_t) = \mathrm{softmax}(W_\pi f_\theta(G_t)), \qquad V_\theta(s_t) = W_V f_\theta(G_t), \tag{3}$$

where π_θ denotes the stochastic policy and V_θ the state-value estimator.

Action masking and feasibility. Only channels that are both *visible* and *operationally feasible* are permitted as actions. A binary mask $M_t \in \{0,1\}^{|\mathcal{V}|}$ encodes this constraint. The masked policy distribution is normalized as:

$$\pi'_\theta(a_t|s_t) = \frac{\pi_\theta(a_t|s_t) \odot M_t}{\sum_a \pi_\theta(a_t|s_t) M_t}, \tag{4}$$

where infeasible actions (self-edges, disconnected nodes, or already-visited peers) are assigned zero probability. This ensures compliance with LN routing rules and keeps exploration within the deployable control surface.

Flow-based reward proxy. Our training and evaluation hinge on s–t Max-Flow (equivalently, Min-Cut) as a reward surrogateWhile full packet-level simulators or stochastic traffic models could yield closer approximations of routing success or latency, they require strong assumptions (arrival processes, path selection rules, fee dynamics, MPP splitting behavior) and are computationally expensive to re-run at each training iteration. In contrast, push–relabel (or equivalent) max-flow solves in polynomial time even on large graphs, allowing us to recompute reward signals per step.

On the fixed budget of 5 actions. We acknowledge that real-world Lightning Network operators may allocate liquidity using larger or even continuous budgets. In early development, we conducted internal sweeps over multiple allocation budgets ($K \in \{1, 3, 5, 10, 20\}$) to identify a configuration that balanced reward signal, stability, and realism. Empirically, $K{=}5$ yielded the most reliable policy gradients and clearest throughput signal:

5 Experimental Details

A full overview of the hardware and experimental setup can be found in App. E

5.1 Data Processing

The data used in this experiment consists of three network snapshots collected from lightning node operated by Amboss Technologies on May 15th, July 15th and October 2nd 2025. These snapshots includes a list of all open channels, their capacities $c_{\{u,v\}}$ as well as the capacities of the associated nodes C_u. Other node and channel features are present but not used in our case. Each channel is first split into directed edges then aggregated for all the edges between the same pair of nodes, summing the capacities. With the directed edges and specified capacities, we then sample the local balances random uniformly. Once we have this list of directed edges and local balances $(y_{(u,v)}, y_{(v,u)})$, we are able to create a weighted directed graph using the iGraph python library [**?**]. For all learning experiments we extract two representative sub-graphs: a 1000 node subgraph for rapid prototyping and

Table 1: MPNN actor–critic configuration.

Parameter	Value
Num node features	4
Num edge features	3
Hidden dimension	64
Num message-passing layers	2

hyper-parameter sweeps, and a 5000 node sub-graph that captures $\approx 90\%$ of total network capacity while discarding the sparse long-tail of tiny nodes. Both sub-graphs are selected by highest-degree ranking to preserve core connectivity.

Baseline Policies. To put the learned policy into context, we compare against three non-learning baselines that require progressively more graph information:

1. **Random**. A uniformly random valid node is chosen at every step.
2. **Degree Centrality**. Nodes are sampled proportionally to their (scaled) out-degree.
3. **Betweenness Centrality**. Nodes are sampled proportionally to their betweenness scores.

All baselines observe the same action mask as the agent, maintaining fairness in feasibility constraints.

Training Protocol. For every training episode the agent executes $k = 5$ actions, gathers 10 trajectories, and then performs one PPO update. Training stops after 250 episodes (< 6 h on the reference GH200). Early stopping is triggered if the average ΔMaxFlow over the last 10 episodes fails to improve for 20 consecutive epochs. A diagram of the training loop can be found in Figure 2 in A. with PPO seetings found in 2

Message passing. We use an edge-conditioned MPNN with a permutation-invariant max aggregator; full equations and ablations are deferred to App. E.1. We use $L = 2$ message-passing blocks, hidden size 64, ReLU activations, and layer normalization after each block; actor/critic heads are linear unless noted. Complete model settings appear in Table 1; the architecture diagram is Figure 1 (App. A).

5.2 Evaluation Protocol and Metrics

For each episode we fix (source, sampled balances, PRNG seed) across methods and measure absolute gain $\Delta F = F^{\text{after}} - F^{\text{before}}$. Primary metric: paired uplift vs. Betweenness (pp). Secondary: absolute ΔF [BTC] and win-rate. Uncertainty is reported as 95% CIs computed over episodes. Budget: k=5 actions at $0.20\,\mathrm{BTC}$ each.

Primary metric. We report *relative maxflow improvement over Random baseline* as the main evaluation criterion. This is measured in percentage points, computed per episode as the relative gain of policy π over Random, then averaged across n=1000 paired episodes with 95% confidence intervals.

Secondary metrics. (i) Absolute improvement ΔF in BTC, averaged across paired episodes; (iii) win rate, the fraction of paired episodes in which π achieves higher max-flow than Betweenness.

Episode budget. Each episode allows k=5 actions (channels) with 0.2 BTC per action (Section 5). Metrics are recorded immediately before the first and after the final action.

6 Results

6.1 Main Results on the 5k Subgraph

On the 5k subgraph (n=1000 paired episodes), the *MP-Flow* model achieves the highest absolute throughput gain with $\Delta F = 0.168 \pm 0.003$ BTC and delivers a significant relative improvement over Betweenness, see Table 3 and Fig. 4 (App. A). Centrality baselines remain strong: Random choice attains $\Delta F = 0.089 \pm 0.003$ BTC and anchors the comparison. All other methods underperform

Table 2: PPO hyperparameters.

Parameter	Value
Clip ratio (ϵ)	0.2
Value loss coefficient (c_1)	0.5
Learning rate (α)	0.002
PPO epochs	5
Max grad norm	1.0

Table 3: Main results on 5k subgraph (n=1000 paired episodes). Values are mean ±95% CI, measured relative to the Random baseline. ΔF is absolute max-flow uplift in BTC.

Policy	Relative Improvement [%]	ΔF [BTC]
Random (BL)	–	0.089 ± 0.003
Degree	96.392 ± 40.877	0.154 ± 0.003
Betweenness	110.245 ± 36.810	0.163 ± 0.003
GCN	104.446 ± 18.817	0.150 ± 0.003
MP-Flow	**119.983 ± 36.023**	**0.168 ± 0.003**

Notes. Relative improvement is computed w.r.t. Random baseline. Dashed lines separate centrality-based and learned policies.

relative to Betweenness on uplift with CIs excluding zero. Figure 4 App. A corroborates the absolute ordering in ΔF, while Fig. 3 highlights the relative improvement margins.

Table 4: Pairwise win rates (%) between policies on 5k subgraph. Cell (i, j) is the fraction of episodes where row i outperforms column j.

	Random	Degree	Betw.	GCN
Degree	96.0	–	48.2	51.1
Betweenness	97.2	51.8	–	58.4
GCN	93.4	48.9	41.6	–
MP-Flow	98.0	60.3	63.2	68.7

6.2 Ablation Study: Graph Scale and Hub Removal

We stress-test along two axes: subgraph size $N \in \{1k, 2k, 3k, 4k, 5k\}$ and targeted hub removal (removing $\{0, 5, 10, 25, 50\}$ highest-capacity nodes), each with n=100 paired episodes. MP-FLOW matches or exceeds Betweenness across sizes—with small gains at 1k–2k, a negligible dip at 3k, and a renewed margin at 4k–5k—and its advantage widens as top hubs are pruned, consistent with bottleneck relief rather than hub chasing (Table 5). Full means and 95% CIs for all methods are in App. B; corresponding plots are in Figs. 5 and 6. Baseline is shown here as the best baseline heuristic from the results of our main study.

6.3 Cross Validation

To evaluate temporal robustness, we train each policy on one Lightning Network snapshot and test on a subsequent (forward) or prior (backward) snapshot, yielding three cross-time pairs: D1→D2, D2→D3, and D3→D1. We added a Graph Attention Network (GAT) baseline after completing the primary experiments because its design is closely related to the GCN/MPNN variants already reported. To keep the main comparison consistent with the originally run baseline set and within the available compute budget (multi-seed × multi-snapshot), we report GAT only in the cross-snapshot generalization study (§6.3) and provide full details in the Appendix. Pilot runs on the seen-graph setting showed similar ordering to the main table and do not alter our conclusions.

MP-Flow achieves the strongest relative improvements on both forward directions (D1→D2: +90.6% ± 5.3, D2→D3: +99.5% ± 6.1), demonstrating consistent generalization as the network evolves over time. On the backward pair (D3→D1), GCN slightly leads (+80.0% ± 4.3) with MP-

Table 5: Ablation summary: MP-FLOW vs Betweenness (best baseline heuristic) (Δ over Betweenness; positive favors MP-FLOW).

	Graph size N				
	1k	2k	3k	4k	5k
ΔF [BTC]	0.024	0.018	−0.002	0.007	0.007
Relative Improvement [pp]	6.77	6.63	−0.64	3.46	4.38
	Targeted hub removals (count)				
	0	5	10	25	50
ΔF [BTC]	0.004	0.005	0.004	0.011	0.007
Relative Improvement [pp]	2.16	3.19	3.07	8.89	7.69

Table 6: Cross-snapshot generalization (n=3 seeds; n=250 paired episodes per cell). Values are mean ±95% CI of relative max-flow improvement (%) over the Random baseline (ratio-of-means; bootstrap CI). Train→Test pairs probe temporal drift (forward/backward).

Policy	D1 → D2 (May → Jul)	D2 → D3 (Jul → Oct)	D3 → D1 (Oct → May)
	Rel. Improvement [%]	Rel. Improvement [%]	Rel. Improvement [%]
Random (BL)	–	–	–
Degree	73.914 ± 5.218	85.786 ± 5.722	66.226 ± 4.176
Betweenness	85.121 ± 4.997	88.411 ± 5.734	77.233 ± 4.474
GCN	76.801 ± 5.257	92.942 ± 5.525	**80.030** ± 4.264
GAT	87.849 ± 5.255	89.204 ± 5.918	60.321 ± 4.357
MP-Flow	**90.558** ± **5.280**	**99.512** ± **6.149**	72.002 ± 4.173

Flow close behind (+72.0% ± 4.2). Among heuristic baselines, Betweenness remains the strongest yet continues to trail the learned policies in every direction. Confidence intervals for heuristics are tight—reflecting stable but lower gains—while learned policies show modestly wider intervals consistent with higher mean uplifts. Overall, these results indicate that the learned models, and MP-Flow in particular, transfer reliably across temporally separated Lightning Network snapshots, with robust forward generalization under topological drift.

6.4 Discussion

Why does MP-FLOW beat GCN and heuristics? We posit three complementary reasons tied to the objective, the aggregator, and the critic.

(i) Objective alignment via RL. Heuristics (Degree/Betweenness) optimize static surrogates that ignore directed balances, per-episode budget constraints, and global flow interactions. By contrast, PPO directly optimizes an estimate of max-flow improvement, using per-step rewards that reflect *marginal* changes in the network's s–t capacity surface. The paired protocol further removes variance induced by the stochastic balance sampler, making the learning signal well aligned with the target metric.

(ii) Max aggregation as a bottleneck detector. GCNs with mean aggregation dilute sharp local evidence: high-capacity, low-fee neighbors are averaged with weak or irrelevant ones, leading to oversmoothing and loss of contrast in precisely the settings where payments are constrained by *bottlenecks*. The MPNN's element-wise max aggregation preserves extreme features in each receptive field, acting as a differentiable "bottleneck lens." This inductive bias is well matched to max-flow, which is itself governed by min-cut structures—i.e., deciding the episode's return often hinges on a few limiting edges/nodes rather than the average neighborhood. Empirically, this manifests in higher win rates and larger paired uplifts.

(iii) Value estimation with global max pooling. The critic's global max pooling focuses the value estimate on the most constraining substructure (e.g., the tightest cut touching the source), improving

credit assignment for actions that relieve the dominant bottleneck. In contrast, GCN baselines aggregate globally by means or sums, which can overweight widespread but *non-binding* connectivity.

A neuro-algorithmic reasoning view. The combination of (a) max-type message passing, (b) a value head that emphasizes the dominant constraint, and (c) an RL loss that rewards marginal flow improvements, encourages the network to learn *algorithmic templates* reminiscent of push–relabel / min-cut reasoning. Messages propagate "capacity potential" and "cut tightness" signals; the actor then selects nodes that either (1) create new disjoint paths, or (2) thicken the current min-cut around the source. In other words, MP-FLOW appears to internalize a coarse, differentiable version of classical flow heuristics—an instance of *neuro-algorithmic reasoning* where learned computation mirrors the structure of the target algorithm while remaining en to end trainable.

Robustness: hubs and scale. Heuristics overweight hubs, so their performance falls when the hub set is pruned. MP-FLOW instead prioritizes *capacity-aware connectivity*, often allocating to medium-degree nodes that unlock near-disjoint routes or thin the active min-cut around the source; max aggregation preserves these few decisive signals rather than averaging them away. With a fixed $k{=}5$ budget, larger N also dilutes global centrality (scores flatten and helpful hubs are rarer for a given source), yet MP-FLOW remains competitive because features are re-normalized per graph, max-based message passing amplifies high-capacity/low-friction corridors near the source, the critic's global max pooling focuses value on the tightest cut, and action masking restricts to feasible peers. Empirically, margins persist under hub removals and re-emerge at $N{=}$4k–5k (Figs. 6, 5).

7 Conclusion

We posed liquidity placement on the Lightning Network as a graph decision process and trained a compact PPO agent with an MPNN-Max backbone to maximize *absolute* max-flow uplift under a fixed per-action budget. On a 5k-node subgraph and 1000 paired episodes, MP-FLOW achieved the largest absolute gain and a statistically significant paired uplift over the strongest heuristic baseline, Betweenness. The policy's advantage persists across graph scales (1k–5k) and under targeted hub removals, suggesting that it leverages capacity-aware bottleneck structure rather than raw centrality.

For operators, this matters because Betweenness is the prevailing industry heuristic for peer selection; exceeding it indicates immediate, practical benefit without requiring payment traces or heavy models. The agent remains lightweight (two message-passing layers, masked policy head) and deployable.

Limitations include the uniform balance sampler, a fixed $k{=}5$ per-action budget, and max-flow as a proxy for realized success rate and economic yield. Nonetheless, results indicate that a simple MPNN with max aggregation, trained end-to-end with PPO, can reliably outperform topology-only heuristics on real LN topology. Future work will incorporate richer reward terms (fees and costs), hybrid discrete/continuous actions, and balance priors, and will evaluate across time on full-network snapshots.

8 Future Work

We view MP-FLOW as an extensible foundation. As well as an intensive hyperparameter grid search, there are many avenues for further development

Richer, simulation-backed rewards. A traffic simulator or replayed-demand model would allow augmenting the reward beyond max-flow uplift to include fee revenue and carrying costs. A scalarized objective could be

$$\begin{aligned} r_t \;=\; & \lambda_{\text{flow}} \left(F_t - F_{t-1}\right) \\ & + \lambda_{\text{fees}} \left(R_t^{\text{fees}} - R_{t-1}^{\text{fees}}\right) \\ & - \lambda_{\text{cost}} \left(C_t^{\text{on-chain}} + C_t^{\text{lease}}\right). \end{aligned}$$

with weights chosen by the operator. This would enable yield-aware allocation policies.

Expanded action space. Beyond discrete peer selection, we plan to (i) add a *continuous resource allocator* that distributes a per-step budget B across chosen peers, e.g. $\alpha = B \cdot \mathrm{softmax}(z)$ to ensure

positivity and budget feasibility, and (ii) introduce operational actions such as *channel closing* and *rebalancing* with friction costs. This yields a hybrid discrete/continuous policy that more closely reflects operator tooling.

References

[1] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993. 1, 2

[2] Vincent Davis, Vikash Singh, and Emanuele Rossi. Channel balance interpolation in the lightning network via machine learning. IEEE ICBC 2025 Workshop: NextGenDLT (Program Listing), 2025. Talk/program reference. 3

[3] Michael Folkson. Plugging into C-Lightning: The Future of Lightning Plugins Is Bright, March 2020. 3

[4] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. 1

[5] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977. 1

[6] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *NeurIPS*, 2017. 1, 2

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proc. ICML*, 2017. 2, 21

[8] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, October 1988. 19

[9] L. R. Ford Jr and D. R. Fulkerson. Maximal Flow Through a Network. *Canadian Journal of Mathematics*, 8:399–404, January 1956. 19

[10] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2017. 2

[11] Lightning Labs. Autoopen. https://docs.lightning.engineering/lightning-network-tools/lightning-terminal/autoopen, 2024. Accessed 2025-08-28. 3

[12] Lightning Labs. Getnodemetrics (betweenness centrality) — lnd api. https://lightning.engineering/api-docs/api/lnd/lightning/get-node-metrics/index.html, 2024. Accessed 2025-08-28. 3

[13] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, July 2019. arXiv:1509.02971. 4

[14] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning, June 2016. arXiv:1602.01783. 4

[15] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2

[16] M E J Newman. *Networks: An Introduction*. Oxford University Press, 2010. 1

[17] Rene Pickhardt. An upper bound for the probability to be able to successfully conduct a payment on the lightning network. GitHub Notebook, 2021. Accessed Aug. 28, 2025. 3

[18] Rene Pickhardt and Stefan Richter. Optimally reliable & cheap payment flows on the lightning network. *arXiv preprint arXiv:2107.05322*, 2021. 3

[19] Rene Pickhardt, Sergei Tikhomirov, Alex Biryukov, and Mariusz Nowostawski. Security and privacy of lightning network payments with uncertain channel balances. *arXiv preprint arXiv:2103.08576*, 2021. 3

[20] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network:. 2

[21] Elias Rohrer, Jason Malliaris, and Florian Tschorsch. Discharged payment channels: Quantifying the lightning network's resilience to attacks. In *IFIP Networking*, 2019. 1

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. arXiv:1707.06347. 1, 2, 3, 4

[23] István Seres, András Gulyás, József Stéger, and András Benczúr. Topological analysis of the lightning network. In *Financial Cryptography and Data Security Workshops*, 2020. 1, 2

[24] Vibhaalakshmi Sivaraman, Shaileshh Venkatakrishnan, Mohammad Alizadeh, Giulia Fanti, and Pramod Viswanath. *Routing Cryptocurrency with the Spider Network*. September 2018. 3

[25] Vincent, Emanuele Rossi, and Vikash Singh. Channel balance interpolation in the lightning network via machine learning. *arXiv preprint arXiv:2405.12087*, 2024. 3

[26] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: Efficient Dynamic Routing for Offchain Networks, June 2019. arXiv:1902.05260. 3

[27] Yifei Yang, Dongmian Zou, and Xiaofan He. Graph Neural Network-Based Node Deployment for Throughput Enhancement. *IEEE Transactions on Neural Networks and Learning Systems*, 35(10):14810–14824, October 2024. Conference Name: IEEE Transactions on Neural Networks and Learning Systems. 3

A Additional Figures

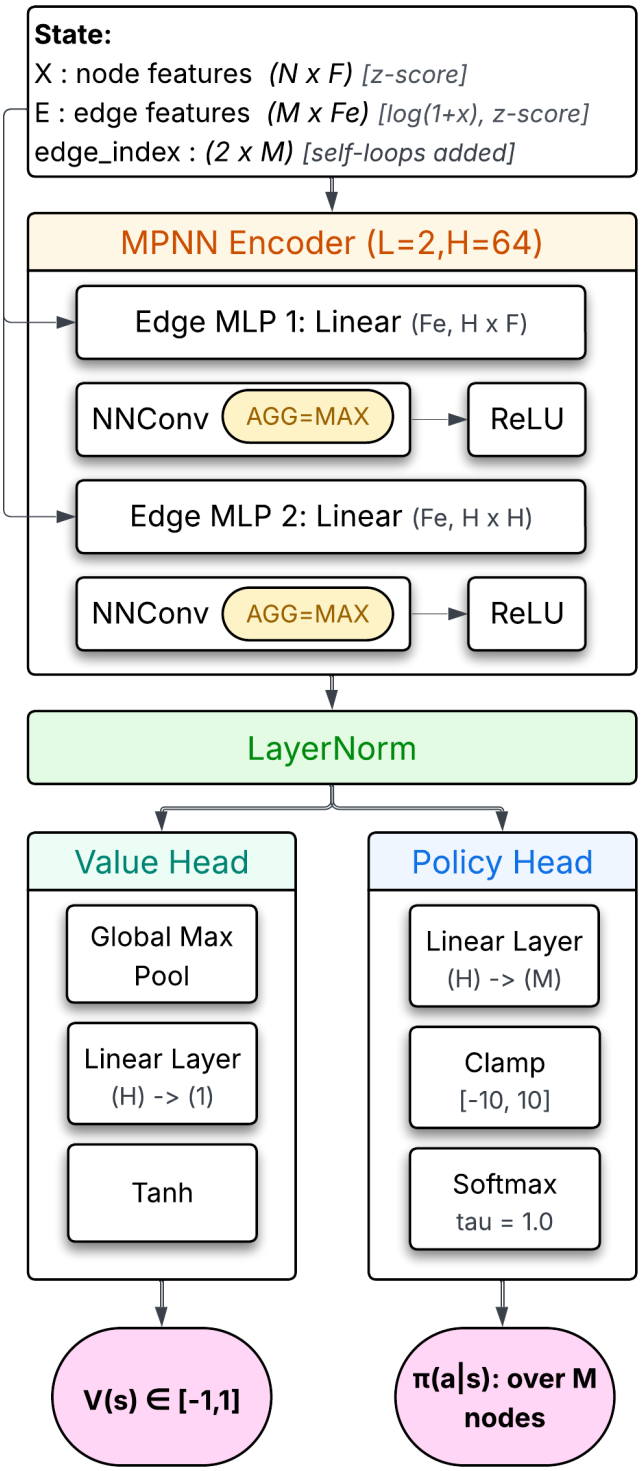

Figure 1: MPNN–Max actor–critic used in HERMES-1. Inputs are node features $X \in \mathbb{R}^{N\times F}$, edge features $E \in \mathbb{R}^{M\times F_e}$ ($\log(1+x)$ then z-score), and the directed edge list (self-loops added). The encoder has two edge-conditioned convolutions (NNConv) whose weights $W(e_{ij})$ are produced by small edge MLPs; messages are aggregated by element-wise *max*, followed by ReLU and a LayerNorm. The actor maps node embeddings through Linear$(H\to 1)$, clamps logits to $[-10, 10]$, and applies a softmax over nodes to produce $\pi(a \mid s)$. The critic applies *global max pooling* over nodes, then Linear$(H\to 1)$ and $\tanh$ to output the state value $V(s) \in [-1, 1]$. H denotes the hidden width.

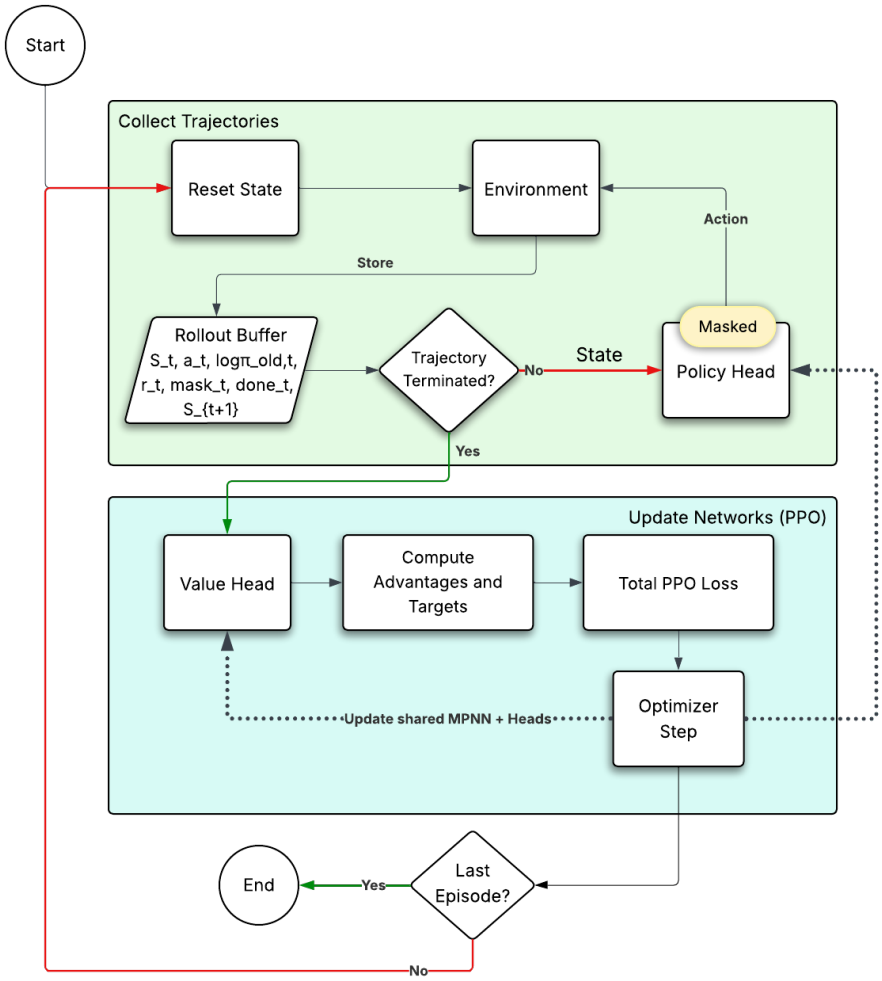

Figure 2: PPO training flow. We collect K on-policy trajectories by sampling actions from a *masked* policy head and append transitions $(s_t, a_t, \log \pi_{\text{old},t}, r_t, \text{mask}_t, \text{done}_t, s_{t+1})$ to the rollout buffer. After collection, the value head (no grad) provides V_t, V_{t+1} to compute TD residuals $\delta_t = r_t + \gamma(1 - \text{done}_t)V_{t+1} - V_t$, advantages $A_t = \text{GAE}(\delta_t; \gamma, \lambda)$, and targets $R_t = A_t + V_t$; A_t is standardized. PPO then runs for multiple epochs and mini-batches, optimizing the clipped policy loss, value MSE, and entropy bonus with gradient clipping and Adam, updating the shared MPNN encoder and the policy/value heads.

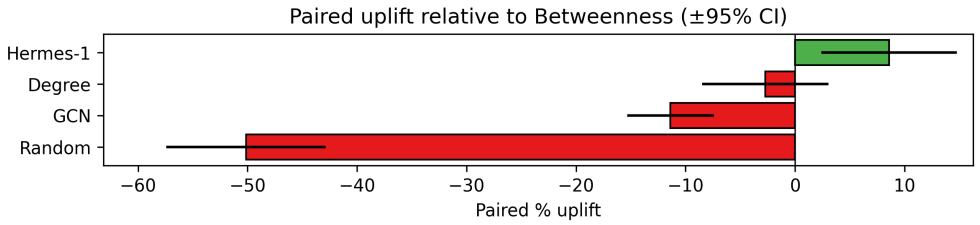

Figure 3: Main results on 5k subgraph (n=1000 paired episodes). Values are mean $\pm$ 95% CI *over per-episode paired differences*; policy weights fixed (single training seed). ΔF is in BTC.

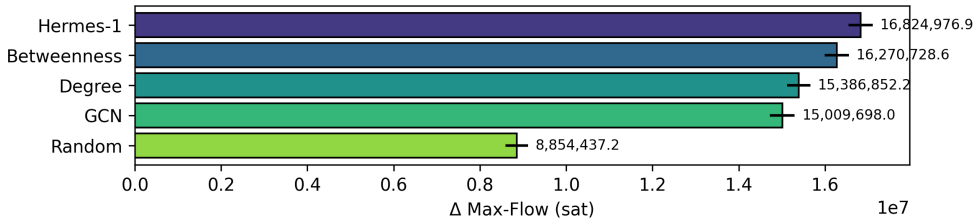

Figure 4: Absolute gain ΔF in satoshis.

B Full Ablations

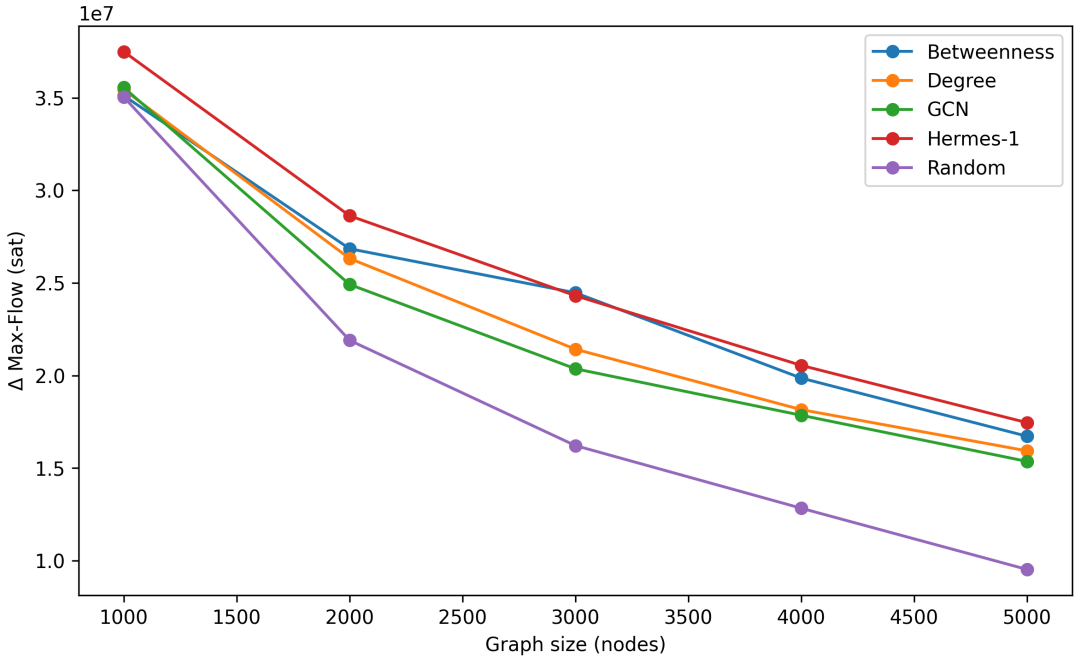

Figure 5: Robustness under graph size. Mean increase in max-flow (sat) after allocations vs. number of nodes included (ranked by degree).

Table 7: Graph size ablation: mean increase in max-flow [satoshi].

Policy	1k	2k	3k	4k	5k
Betweenness	35,119,967	26,847,743	24,467,038	19,856,956	16,715,054
Degree	35,.442,061	26,320,441	21,418,223	18,152,661	15,925,096
GCN	35,557,096	24,916,419	20,360,919	17,848,011	15,354,765
Hermes-1	**37,498,380**	**28,628,957**	**24,310,135**	**20,543,831**	**17,447,526**
Random	35,042,641	21,898,126	16,210,424	12,824,566	9,516,612

Table 8: Graph size ablation: 95% CIs on mean [satoshi].

Policy	1k	2k	3k	4k	5k
Betweenness	4,700,479	2,853,931	1,454,478	1,034,191	823,694
Degree	3,570,816	2,411,171	1,861,081	1,233,102	799,558
GCN	3,637,812	2,306,042	1,610,266	1,289,970	952,225
Hermes-1	**3,584,004**	**2,638,046**	**1,709,296**	**1,207,376**	**845,026**
Random	3,378,914	2,194,258	1,524,510	1,129,366	845,773

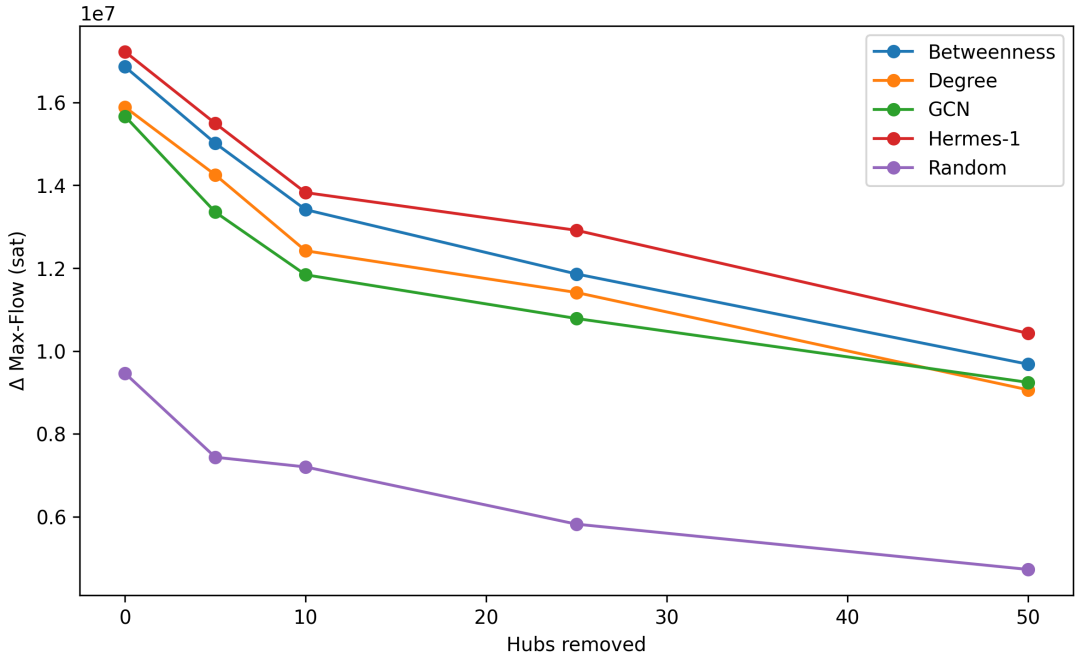

Figure 6: Robustness under targeted hub removal (5k subgraph). Mean increase in max-flow (sat) after allocations vs. number of highest-degree hubs removed.

Table 9: Targeted hub removal: mean increase in max-flow [satoshi].

Policy	0	5	10	25	50
Betweenness	16,863,171	15,016,857	13,413,550	11,860,529	9,682,913
Degree	15,889,267	14,253,300	12,421,288	11,413,222	9,064,480
GCN	15,668,308	13,354,329	11,841,622	10,785,778	9,244,112
Hermes-1	**17,227,615**	**15,496,340**	**13,825,515**	**12,914,576**	**10,427,826**
Random	9,465,738	7,437,863	7,203,007	5,821,940	4,728,582

Table 10: Targeted hub removal: 95% CIs on mean [satoshi].

Policy	0	5	10	25	50
Betweenness	825,513	578,908	637,288	486,358	508,064
Degree	783,643	666,313	788,502	455,548	503,849
GCN	849,652	648,058	709,782	519,346	514,983
Hermes-1	**812,871**	**611,489**	**812,458**	**389,750**	**472,074**
Random	830,090	656,729	651,901	536,008	483,645

C Definitions and Implementation Details (for reference)

Feature definitions. Node features: PageRank, capacity ratio, normalized degree, local clustering; z-scored per reset. Optional edge features: base fee (msat), fee rate (ppm), channel capacity (sat) with $\log(1+x)$ then z-score.

Training details. Two message-passing layers (hidden width 64), **max** aggregation. PPO with masked discrete actions; critic uses global max pooling. Additional setup (optimizer, epochs, early stopping) as in the main text; full diagrams are in Figs. 1–2.

D Max-flow: full formulation

The Maximum Flow (Max-Flow) problem is a fundamental concept in network flow theory, first introduced by Lester R. Ford Jr. and Delbert R. Fulkerson in 1956. Their seminal work laid the foundation for analyzing flow networks by formulating the Max-Flow Min-Cut theorem and proposing the Ford-Fulkerson algorithm for computing the maximum flow in a network [9]. It involves determining the maximum amount of flow that can be sent from a source node s to a sink node t in a flow network, represented as a directed graph $G = (V, E)$ where V is the set of vertices (nodes) and E is the set of edges (links). Each edge $(u, v) \in E$ has a non-negative capacity $c(u, v)$ representing the maximum flow that can pass through that edge. The problem is subject to the following constraints:

- **Capacity Constraints**: For every edge (u, v), the flow $f(u, v)$ must satisfy:

$$0 \leq f(u, v) \leq c(u, v)$$

- **Flow Conservation**: For every node u except the source s and sink t, the sum of flows into u equals the sum of flows out of u:

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

The objective is to maximize the total flow from the source to the sink:

$$\text{Maximize} \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

Over the years, several algorithms have been developed to solve the max-flow problem more efficiently. One such algorithm is the Push-Relabel algorithm, also known as the Preflow-Push algorithm, introduced by Andrew V. Goldberg and Robert E. Tarjan in 1986 [8]. The Push-Relabel algorithm improves upon previous methods by utilizing a different approach that locally adjusts flows and maintains a preflow—a flow that allows excess at intermediate nodes—to find the maximum flow more efficiently, especially in dense networks.

E Experimental Setup

Experiments ran on an Intel 12$^{\text{th}}$-Gen Core i7 with 64 GB RAM, an RTX 3090 (24 GB) and a GH200; code is in Python 3.11 using PyTorch 2.2, PyTorch-Geometric 2.5, NetworkX 3.3, and iGraph 0.10, with custom RL utilities following the `gymnasium` API. We use a public LN snapshot from 16 July 2025 (7,691 nodes; 37,018 bidirectional channels), converted into a directed, weighted graph as in Sec. 4.1; only total channel capacities are observed, so per-direction balances are sampled once per run, and node/edge features are z-scored per reset. The `LightningNetworkEnv` extends a base graph environment: each episode permits k=5 masked liquidity actions (open/top-up); transitions deterministically update capacities; and the per-step reward is the marginal change in absolute max-flow, $r_t = F_t - F_{t-1}$, computed via push–relabel. The total maxflow reward is the sum of total flow computed a list of 50% of the nodes in the network, sampled randomly as targets from our fixed source.

$$S \sim \text{Uniform}\big(\{\, U \subseteq V : |U| = \tfrac{|V|}{2} \,\}\big)$$

Episodes end after the fifth action or an invalid selection; repeated max-flow solves make training largely CPU-bound. For each undirected channel (u, v) with total capacity C_{uv}, we draw a random proportion $\alpha_{uv} \sim \text{Uniform}(0, 1)$. We then assign $\alpha_{uv}C_{uv}$ to the directed edge $u \to v$ and the remainder $(1 - \alpha_{uv})C_{uv}$ to the reverse edge $v \to u$. This procedure ensures that the two directed balances c_{uv} and c_{vu} are random but always sum exactly to the channel capacity C_{uv}, providing an unbiased split of liquidity between the two directions. Each episode, this sampling is repeated.

$$\begin{aligned} &\alpha_{uv} \sim \text{Uniform}(0, 1), \\ &c_{uv} = \alpha_{uv}\, C_{uv}, \qquad\qquad c_{uv} + c_{vu} = C_{uv}. \\ &c_{vu} = (1 - \alpha_{uv})\, C_{uv}, \end{aligned}$$

Uncertainty and significance. Unless stated otherwise, confidence intervals are $\pm 95\%$ CI on means: $\bar{x} \pm 1.96\, s/\sqrt{n}$ where x is either ΔF or d and s its sample standard deviation.

Implementation Notes. The MPNN backbone is implemented with `torch_geometric.nn.MessagePassing`. Edge attributes are concatenated channel capacity and balance ratio; self-loops are added automatically. Code and trained checkpoints are available in the accompanying repository.

For each reset of the environment we build two matrices: a node–feature matrix $X \in \mathbb{R}^{N\times 4}$ and (optionally) an edge–feature matrix $E \in \mathbb{R}^{M\times 3}$. All features are re-computed after any topological change and standardised (z-score per dimension) before being fed to the MPNN.

Environment. We implement a lightweight `BaseNetworkEnv` (graph ops via NetworkX) and two specializations: `RandomNetworkEnv` (Barabási–Albert graphs) for smoke tests and `LightningNetworkEnv` for real LN snapshots (loading, masked open/top-up actions, deterministic capacity updates). The API mirrors `gymnasium` with custom `reset/step`; rewards use push–relabel as defined earlier, enabling drop-in swapping of graph sources while keeping identical RL code.

E.0.1 Node Features

1. **PageRank Centrality** $C_{PR}(u)$. Captures global influence and replaces the earlier adjacency indicator:
$$X_{u,1} = C_{PR}(u).$$
2. **Capacity Ratio** $C_{CR}(u)$. Fraction of total network liquidity held by u:
$$C_{CR}(u) = \frac{C_u}{\sum_{v\in V} C_v}, \qquad X_{u,2} = C_{CR}(u).$$
3. **Normalised Degree** $C_D(u)$. Relative connectivity of u:
$$C_D(u) = \frac{\deg(u)}{N - 1}, \qquad X_{u,3} = C_D(u).$$
4. **Local Clustering Coefficient** $C_C(u)$. Density of the node's neighbourhood:
$$X_{u,4} = C_C(u).$$

When `use_edge_features` is enabled the MPNN receives three attributes per directed edge (u, v):

1. **Base Fee** f_{uv}^{base}. Fixed cost charged on any payment through (u, v):
$$X_{uv,1} = f_{uv}^{\text{base}}\,[\text{msat}].$$
2. **Fee Rate** f_{uv}^{rate}. Proportional fee charged per unit of forwarded liquidity:
$$X_{uv,2} = f_{uv}^{\text{rate}}\,[\text{ppm}].$$
3. **Channel Capacity** c_{uv}. Total liquidity available on channel (u, v):
$$X_{uv,3} = c_{uv}\,[\text{sat}].$$

To mitigate heavy-tailed distributions we first apply a $\log(1 + x)$ transform:
$$\tilde{e}_{uv,i} = \log\bigl(1 + e_{uv,i}\bigr) \quad (i = 1, 2, 3),$$
followed by per-column normalisation $E \leftarrow (\tilde{E} - \mu_E) \oslash (\sigma_E + \varepsilon)$.

Edges with missing fee information default to zeros, ensuring that the feature tensor always has shape $M \times 3$.

E.1 MPNN-Max Architecture

The actor and critic share the same message-passing backbone but have separate output heads.

1. **Input**. Node feature matrix $X \in \mathbb{R}^{N \times F}$ and edge feature matrix $E \in \mathbb{R}^{M \times F_e}$.
2. **Message Passing Block** (repeated $L = 2$ times). For each edge $e = (u, v)$ the message function is
$$m_{uv}^{(l)} = \sigma\big(W_m^{(l)}[h_u^{(l)} \,\|\, h_v^{(l)} \,\|\, e_{uv}]\big),$$
followed by node update
$$h_u^{(l+1)} = \sigma\Big(W_h^{(l)}\big[h_u^{(l)} \,\|\, \mathrm{AGG}\{m_{vu}^{(l)}\}_{v \in \mathcal{N}(u)}\big]\Big),$$
where AGG is element-wise *maximum*. Max aggregation proved empirically more discriminative for heterogeneous channel capacities than mean. This design follows the Message Passing Neural Network (MPNN) framework [7], with edge-conditioned convolutions in the spirit of ECC/NNConv [**?**].
3. **Policy Head (Actor)**. A linear projection maps $h^{(L)}$ to logits $o \in \mathbb{R}^N$. The masked soft-max produces the stochastic policy:
$$\pi_\theta(a_i \mid s) = \frac{\exp(o_i)}{\sum_{j \in \mathcal{A}(s)} \exp(o_j)}, \qquad i \in \mathcal{A}(s),$$
where $\mathcal{A}(s)$ is the valid action set given by the environment mask.
4. **Value Head (Critic)**. Global **max** pooling aggregates node embeddings: $h_G = \max_{u \in V} h_u^{(L)}$. A single fully-connected layer with $\tanh$ activation outputs the state-value estimate $V_\phi(s) \in [-1, 1]$.